

SINCLAIR ZX 8K ROM* UPGRADE

Thomas J. Bent

9016 Flicker Place

Columbia, MD 21045

* Use of the 8K operating system has been granted by Sinclair Research, thereby allowing this upgrade to be possible. I am grateful for their generosity.

Copyright 1981 Sinclair Research
Copyright 1985, T. Bent

February 24, 1985

Dear Sinclair Enthusiast,

Here is the eprom upgrade for your operating system. Version 10. is the current issue. Several revisions have been made to both "fix" and improve the Sinclair BASIC operating system. I'm quite sure that all of the changes will delight you. You will probably forget that it is installed in a few short days. ALL of the software that would run before will still run, but you must have at least 16K RAM. The machine will initialize with 1 or 2K RAM, because of the partial decoding in the machine. Programs will not run though.

There are a few key changes that will remind you that you have indeed upgraded. The single most noticeable change is the automatic FAST edit. Now, whenever you enter an empty cursor, or key in a program line, the computer will jump into FAST mode and stay there until you input SLOW again. If your program requires SLOW mode to work properly, then remember to input SLOW in the direct mode before RUNNING it. This alleviates that annoying "rolling screen" that can't decide which line should be listed first.

The second major change is the FAST initialization. Although the computer still only initializes 16K on power up, even a 64K NEW (POKE 16388,255 POKE 16389,255) will give a K cursor in the lower left corner in about 2 seconds. This routine will not adjust for your LACK of ram. If you only have 16K of ram, then don't poke ramtop up to 64K. I guarantee a crash. To run properly in 2K, POKE 16389,72 and NEW.

The Load routine has been changed slightly also. Before, if you had a bad load, the computer would jump into the middle of the initialization routine and reset the display file and stack pointers, but would leave clutter in memory (not really a problem) and not reset all of the operating system variables. Now, it jumps to the beginning of the NEW command and properly resets everything below RAMTOP. (RAMTOP is still a safe byte. NEW does not affect it.)

The CLS routine has been shortened and modified so as to not collapse the display file anymore. Before, everytime you enter a BASIC line the computer would check to see if you had enough memory to create a machine stack and steal memory from the display file if needed. Then, after computation, the display would have to be regenerated. As you pushed the limits of your memory, you stood a good chance of crashing because of all the stack manipulation. Now, the computer won't take a line unless it can handle it properly. As you approach the limit of your memory you will find that the computer will not take a line with a number, although it will take other lines. There just isn't enough room to compute those big floating point numbers. Try the following listing on a 2K machine to see what I mean:

In the direct mode, DIM A(10)

```
1 REM
10 PRINT
20 PRINT 10
```

The problem of letting the display file overlap the 32K mark still exists, so if you have 64K beware! This problem exists because of the way that the Sinclair handles address line A15 (it is used as a sort of memory map line for the display). If you encroach on address 32768, you crash.

With a fixed size display file, another change is possible. SCROLL is now a very useful (and fast) routine. Before, if you scrolled your screen 22 times and then CLS, you could practically take a nap waiting to regain control of your Sinclair. Not anymore! Now, Flight Simulator acts properly after a crash (literally of course).

As a sideline to having a fixed display file, all those ROM checks to see if the display location is available fall through immediately, thereby slightly speeding up the print routines, such as, TAB and AT.

Do you use a large database? Now you can DIMension large single arrays up to 47872 (BB00h). If you have a monitor program in the 8-16K block and need no BASIC lines, then you can go just over 48000 (BBFFh) with your array. You can't make an array too large for your memory though. You will get a very thoughtful error 4.

Two bugs that have been squished (in the TS1500 also) are the LPRINT and divide bugs. The divide bug is a problem when doing repetitive math work, such as matrix inversion, simultaneous equations, TAN (the computer generates the TAN function by calculating SIN/COS), etc. If you do this type of work, you can notice a reduction in the sum of the squares calculations (see more info on this in SWN vol 1). For an interesting demonstration of this bug, run the short test listing before you put in the new eprom, and put the print line inside the loop (it won't run very long). This will answer the question; "how much is one bit?"

The other obvious bug is the LPRINT bug. This annoying creature sneaks up on you and prints garbage on your printer regardless of the interface or printer that you are using. This one got by Sinclair Research because there was no printer available when they finalized the last issue of the ROM. (Don't you just love aftermarket support!) The problem occurs when you try to print variable numbers less than 0.1 and greater than 1E-5. All of the leading zeroes come out as trash. To get around this problem you have had to convert your numbers to strings and LPRINT the strings. If you LET X = .0001, and then LPRINT X, you will see what I mean. This is no longer a problem.

A few cosmetic changes have made in the character generator also. Because the display on your TV is probably not the best in the world, we have changed some of the bit patterns in order to improve the readability of a few of the characters. The Q, W, V, K and British Pound sign have been modified. The most noticeable change is the pound sign, which is now an apostrophe. Now, the Q, O and 0 are all distinct; and the indeterminate W, U and V are well defined. They will all print that way on the 2040 and ZX printer too. Although the bit patterns are in the eprom, not all of the characters are available to change. If you look in the appendix of your instruction manual, only the first 64 characters are at your disposal (up to Z). The rest of the graphics, inverse characters, tokens and composite characters (" and **) are created by the Sinclair logic chip and the token tables in the ROM. However, any of the first 64 characters can be changed (by further changing the eprom).

The last and most unique change that has been made to date is the modified LPRINT command. This command is transparent until you invoke it. This routine is called by RANDing an address in memory that you want to go to, POKE'ing 16393,1 (VERSN, which is the first byte saved in your program) and LPRINT'ing. For example:

```
10 RAND xxxxx (any address at which you have a working machine code
subroutine: end with RET)
20 POKE 16393,1 (or any odd number)
30 LPRINT (or LPRINT X, LPRINT "HELLO")
```

This is very similar to USR, except that LPRINT has syntax checking and has the power to easily pass variables or text to your routines without a lot of overhead or searching for your data. It can also act just like a USR call, except that you need not return a value, such as, LET X = USR nnnn.

To turn off this command, POKE 16393,0 (or any even number). In machine code use FD3509, which is DEC (IY+9). You can also use INC (IY+9) or RES 0, (IY+9). It's your choice. This byte is saved with your program and RAND USR calls could present a problem. It is a good idea to initialize this in a subroutine when you use it. Entering a program line will not invoke this command, however a direct command without a line number will, so take care.

I am in the process of writing a driver that will link (hopefully) relocatable subroutines together and actually extend the Sinclair Basic operating system. I will let you know when I have something worthwhile. I have a few things in mind, but I am open for both suggestions and submissions. Unfortunately, my duties at SyncWare News prevent me from spending all the time on this project that I would like to put in on it. I do hope that you enjoy it though.

INSTALLATION INSTRUCTIONS:

BEWARE OF STATIC ELECTRICITY

Turn your computer over and remove the 5 small screws holding it together. Remove the back and unscrew the 2 screws holding the PC board to the top half. Gently turn the PC board over exposing the chip side of the board. (Be careful with the keyboard connector. Don't kink it.) Locate the ROM. It is the one that is too small for its socket. Pry it up with a long thin screwdriver. Insert the eprom and its socket in the ROM's place. Make sure that all the socket pins are seated properly before you firmly press the sockets together. (If you break a pin, believe me, soldering those little jumper wires is a real bear!) Close up your case, run it and forget it!

The circuit will fit well in a TS1000, but may not fit in a ZX81 depending on how old it is (due to a redesign of the board). A single socket may be used, but this requires soldering on the eprom. The Eprom may also be used in a 1500, but it again requires soldering on the eprom. It has come to my attention that there are some TS1000's that have the ROM soldered in place. Don't worry. Just clip it out with some small wire cutters. Get one of those blue, suction type desolders (Radio Shack) and clean up the board. You can solder in the socket circuit or get a 28 pin low profile socket and solder it in. Plug in the EPROM and run it! If you have any questions or trouble, drop me a line or call me in the evenings at 301-730-7187.

ZX-81 EPROM UPGRADE CHANGES

```

0000 03FD      OUT FD,A
• 0002 210080   LD HL,8000
• 0005 03C903   JP INIT
0008 2A1540   ERRO LD HL,(CHADD)
000B 221840   LD (X PTR),HL
000E 1845      JR ERRO
0010 A7        INPR AND A
0011 02F107   JP NZ ARCH
0014 03F507   JP PRSP

```

```

034C 0E01      LDBY LD C,01
034E 0600      LNXB LD B,00
0350 3E7F      AGIN LD A,7F
0352 DBFE      IN A,FE
0354 D3FF      OUT FF,A
0356 1F        RRA
0357 3049      JR NC LFAL
0359 17        RLA
035A 17        RLA
035B 3628      JR C LBIT
035D 10F1      DJNZ AGIN
035F F1        POP AF
0360 BA        CP D
• 0361 3063      NOLD JR NC BOLD
• 0363 00        NOP
0364 62        LD H,D
0365 6B        LD L,E
0366 CD4C03     NMIN CALL LDBY

```

```

0303 0DE702     NEW CALL TFAS
• 0306 2A0440     BOLD LD HL,(RAMTP)
• 0309 54         INIT LD D,H
• 030A 5D         LD E,L
• 030B 3E3F      LD A,3F
• 030D 0B        DEC HL
• 030E 3600      CLER LD (HL),00
• 030F 0B        16K- DEC HL
• 03D1 8C        CP H
• 03D2 00FA      JR NZ CLER
• 03D4 2B        EX DE,HL
• 03D5 1810      JR MORE
• 03D7 CDA60D     LINE CALL FL37
• 03DA 2304      CHEK JR Z NOST
• 03DB FDCB0946   FLAG BIT 0,(VERSN)
• 03DE CACB0A     NOST JP Z LPRN
• 03E0 DA3240     YEST LD HL,(SEED)
• 03E3 89        JUMP JP (HL)
• 03E7 2A0440     MORE LD (RAMTP),HL
03E9 00         INIT DEC HL
03EB 363E      LD (HL),3E
03ED 7B        DEC HL
03EF 7B        LD SP,HL
03F0 2B        DEC HL
03F1 220240     LD (ERRSP),HL
03F4 3E1E      LD A,1E
03F6 8D047      LD I,A
03F8 8D06      IM 1
03FA FD210040   LD IY,ERRNR
03FE FD363B40   LD (CDPLG),40
0402 217040     LD HL,4070
0405 220040     LD (DFILE),HL
0408 0619      LILO LD B,19
• 0409 CD570A     CALL MAKE
040D 221040     LD (VAR3),HL
0410 CD9A14     CALL CLER
0413 CDAD14     BASI CALL K-EL
• 0416 CD230F     UAPP CALL FAST
0419 CD2A0A     CALL CLS
041C 2A0A40     LD HL,(S APP)
041F ED5B2340   LD DE,(S TOP)
0423 A7        AND A
0424 ED52      SBC HL,DE

```

;0002 Reflects a preset Ramtop (a la 1500)
;0005 Jump to the proper INIT point

Version 10. Thomas Bent

;0361 This change changes where you go in case of a BaD Load. This location was changed primarily to make more consecutive space in the INIT routine. However, it does make this routine function properly.

;0363 This NOP clears the garbage left by the change from the 3 byte (jump) to the 2 byte (jump relative) command

;03C3 This INIT routine is completely re-written in order to both speed up and add other changes. BC is no longer used and therefore contains 0000 when not in use (instead of a number near ramtop).

; The memory check is no longer present, so you must have at least 16K in order to function properly. A 2K machine will initialize and take BASIC commands, but as soon as you over-write your phantom stack pointer, good-bye. (This is due to partial decoding and repeating of memory segments in a 2K machine.)

;03D7 This is the new location of LPRINT. First you check to see if Syntax is being tested, by checking BIT 7 of Flags. If you are entering a line, then you go to the regular LPRINT routine. If you enter a direct command or are running a program, then you test BIT 0 of VERSN. If it is 0 then you again jump to the LPRINT routine. If it is 1, then you get the number set by RAND and jump to that location. There is no commercial software other than the AERCO printer interface that uses this byte (VERSN) That I know of. They do not use BIT 0 though.

;040A Make the display. This routine was relocated in order to make more space above.

;0416 This one byte change serves a double purpose. It completes initialization in fast mode, and everytime you key in an empty cursor or enter a line you come back in FAST mode.

```

0A2A 0618 CLS LD B,18
0A2C FDCB018E CLLN RES 1,(FLAGS)
0A30 0E21 LD C,21
0A32 C5 PUSH BC
0A33 CD1809 FULL CALL SEDF
0A36 C1 DIS- POP BC
0A37 FDCB3AFE PLAY SET 7,(SPOS3L)
0A3B AF XOR A
0A3C CDF507 CALL PRSP
0A3F 2A3940 LD HL,(SPOS3N)
0A42 7D LD A,L
0A43 B4 OR H
0A44 E67E AND 7E
0A46 20F3 JR NZ,0A3B
0A48 C31809 DONE UP SEDF
0A4B 0620 SCRL LD B,20
0A4D AF PT2- XOR A
0A4E 2B DEC HL
0A4F 77 LD (HL),A
0A50 10FC DJNZ 0A4E
0A52 012103 LD BC,0321
0A55 18F1 JR DONE
0A57 3676 MAKE LD (HL),76
0A59 23 DIS- INC HL
0A5A 10FB PLAY DJNZ MAKE
0A5C 09 RET
0A5D CD170A MCLE CALL ASUB
0A5E 05 HMCL PUSH BC
0A61 78 LD A,B
0A62 2F CPL
0A63 47 LD B,A
0A64 79 LD A,C
0A65 2F CPL
0A66 4F LD C,A
0A67 03 INC BC
0A68 CDAD09 CALL ADPT
0A6B EB EX DE,HL
0A6C E1 POP HL
0A6D 19 ADD HL,DE
0A6E D6 PUSH DE
0A6F EDB0 LDIR
0A71 E1 POP HL
0A72 09 RET
0A73 2A1440 ELNR LD HL,(ELINE)

```

```

0C0E 2A0C40 SCRL LD HL,(DFILE)
0C11 23 INC HL
0C12 83 PUSH HL
0C13 112100 LD DE,0021
0C15 10 ADD HL,DE
0C16 01 POP DE
0C18 0615 LD B,15
0C19 05 PUSH BC
0C1B 012000 LD BC,0HAD
0C1C EDB0 LDIR
0C1D 13 INC DE
0C1E 03 INC HL
0C1F 01 POP BC
0C20 10F5 DJNZ 0C1A
0C21 2B DEC HL
0C22 034B0A UP SOL3
0C23 80 OFFT ADD A,E
0C24 3D ADD A,L
0C25 2D DEC L
0C26 7F LD A,A
0C27 61 ADD A,C
0C28 49 LD C,C
0C29 0F RRCA
0C2A 00 NOP
0C2B 23 INC HL
0C2C 0F RRCA
0C2D 00 NOP
0C2E 09 LD L,C
0C2F 03 EX AF,AF
0C30 06 DEC B
0C31 D7 #NEW RST 10H
0C32 03 SPOT INC BC
0C33 03 INC BC

```

;0A2A The CLS routine has been truncated to make more space for the SCROLL routine. It no longer collapses the display for any reason. Throughout each of the display and pointer setting routines, there are tests to see if room is available. Since there always is, all of the tests fall through and the machine runs slightly (though not noticeably) faster.

;0A4B This is the last part of the SCROLL routine. It does what the old routine did not. It pads out the line with blanks.

;0A52 This value sets the print position to the lower left the same as "PRINT AT 21,0;" (LD BC 2100 and jump to the print at routine), but since the print at routine leaves 0321 in BC, why not bypass it and speed up the routine that much more.

;0A57 This is the create an empty screen routine. It was put here to make space elsewhere. It is only called from NEW. After the first CLS command (which is always done in FAST mode now), you then have a full display file and it stays that way.

;0C0E The SCROLL command is completely different, and in fact much faster than the old one. It is a modification of a program by Dan Tandberg, called Fast Scrolling (a collection of which is by the way, available in a listing from T. Woods). It is quite good.

;0C29 This is the beginning of the offset table. All of the BASIC command addresses are located in this table. If you relocate (permanently) a given BASIC command, then just put the starting address in the slot designated for the command that you change. That is precisely what we have done with LPRINT. The address 03D7 corresponds to the new routine address. We take over before any flags have been set or variables changed, AND more importantly, we do not touch the old routine at all.


```

1409 CD1011 DIM CALL LUVA
140C C29A0D DCER JP NZ CERR
140F CDA60D CALL FL37
1412 2008 JR NZ DRUN
1414 CBB1 RES 8,C
1416 CDA711 CALL STVA
1419 CD1D0D CALL CSEV
141C 3808 DRUN JR C DLTR
141E C5 PUSH BC
141F CDF209 CALL NXBV
1422 CD500A CALL MMCL
1425 C1 POP BC
1426 CBF9 DLTR SET 7,C
1428 0600 LD B,00
142B C5 PUSH BC
142D 210100 LD HL,0001
142E CB71 BIT 6,C
1430 2002 JR NZ DNUM
1432 2E05 LD L,05
1434 EB DNUM EX DE,HL
1436 E7 DMLO RST 20H
1438 26BB DM48 LD H,BB
1439 CDD012 CALL INBC
143B DA3112 JP C GERR

```

;1436 This is one of those oversight routines that is useless. Sinclair must have thought that no one would ever use the ZX81 with more than 16K. He put an arbitrary limit on the size of a single array at 16K. I changed this LD H,40 to LD H,BB, but it may as well been FF. You can't make an array larger than memory available anyway. Other routines watch out for this error. This routine could be eliminated and something else put in that is more useful. There is at least 8 bytes here.

```

1688 3E10 ZSTR LD A,10
168A D7 RST 10H
168B 10FB DJNZ ZSTR
168D 1809 JR PTM
168F 3E10 ZEXP LD A,10
16C1 D7 RST 10H
16C2 C5 TINT DEC (HL)
16C3 34 INC (HL)
16C4 E8 RET PE
16C5 3E18 LD A,18
16C7 D7 RST 10H
16C8 C5 PTM DEC (HL)
16C9 34 INC (HL)
16CA E8 RET PE
16CB CDD018 CALL PDGT
16CD 18F8 JR PTM
16CE 7E PDGT LD A,(HL)
16CF E60F AND 0F
16D0 CDEB07 CALL PROCD
16D2 2B DEC HL
16D3 09 RET
16D5 7E PADD LD A,(HL)

```

;16BB This oversight originally jumped back only to 16BA and printed whatever was in A (garbage). This change was originally reported in Syntax, and this change has been incorporated in the 1500 ROM.

For more information on the Sinclair ROM, I suggest purchasing Ian Logan's ZX81 ROM Disassembly part A and B. For easy study, enjoyment and other Sinclair related pleasures, I suggest purchase of Hot Z II (although I still prefer Hot Z I) and get the ROM name file. This listing is done with this name file in memory and as you can see, it makes it quite easy to see what is going on (especially with a hardcopy listing and explanation in front of you). It is available from Tom Woods.

```

18AB 20 DV3B ADD HL,HL
18AC 00 EXX
18AD ED6A ADD HL,HL
18AF 00 EXX
18B0 3810 JR C NORE
18B2 ED52 SBC HL,DE
18B4 00 EXX
18B5 ED52 SBC HL,DE
18B7 00 EXX
18B8 300F JR NC DGOS
18BA 19 ADD HL,DE
18BB 00 EXX
18BC ED6A ADD HL,DE
18BE 00 EXX
18BF 17 AND A
18C0 1808 JR DDOU
18C2 ED77 AND A
18C3 ED52 SBC HL,DE
18C5 00 EXX
18C6 ED52 SBC HL,DE
18C8 00 EXX
18C9 37 DGOS GOF
18CA 04 DDOU INC B
18CB 7AA218 JP M DVLO
18CD 78 PUSH AF
18CE 20DA JR NZ DV3B
18D1 00 LD B,A
18D2 00 LD D,C
18D3 00 LD E,C

```

- ☐ The other changes are all in the bit patterns
- ☐ of the letters. For more details on how to
- ☐ change bit patterns, I direct you to SyncWare
- ☐ News, Issue 2/2, and John Oliger's, Video
- ☐ Upgrade ROM changes article.

;18CF This jump originally jumped back to 18BA instead of 18AB. Somebody must have written this routine on overtime, but it is an easy oversight indeed. This is also fixed on the 1500.

TRY THIS. SHOW IT TO YOUR TS
FRIENDS.

QQQ VVV WWW UU OO KKK '''

ALSO TRY POKE 16389,255 ENTER
NEW

DIM A\$(45000)

LET A\$(45000)=1

PRINT A\$(45000)

YOU ARE READY FOR A BIG ZX/PRO-
FILE NOW

1

0.1

.01

.001

.0001

.00001

1E-6

1E-7

1E-8

1E-9

10 LPRINT ''' "TRY THIS. SHOW
IT TO YOUR TS' FRIENDS."

12 SLOW

14 LPRINT "QQQ VVV WWW UU OO
KKK ''' "ALSO TRY POKE 16389
,255 ENTER NEW", "DIM A\$(45000
)"

15 LPRINT "LET A\$(45000)=1 ",
"PRINT A\$(45000)"

16 LPRINT "YOU ARE READY FOR A
BIG ZX/PRO- FILE NOW"

19 LET X=10

20 FOR I=1 TO 10

30 LET X=X/10

40 PRINT X

50 LPRINT X

60 NEXT I

69 PRINT " INPUT ANY KEY"

70 PAUSE 4E4

80 CLS

90 FOR J=1 TO 10

100 FOR I=1 TO 22

110 PRINT "XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXX"

120 NEXT I

130 FOR I=1 TO 22

140 SCROLL

150 PRINT "Y"

160 NEXT I

170 CLS

180 NEXT J

200 LLIST 10

10 LET X=TAN (PI/4)-1

30 FOR I=1 TO 1000

40 LET X=X+X

60 NEXT I

70 PRINT X

90 PRINT I

HARDWARE PROJECTS

Run TS1000 Machine Code in High Memory

John Oliger

ZX81ers don't despair! This next trick was originally published in Syntax Quarterly, Summer 1983. It will allow you to use the 32-48K RAM area for machine code, if you have the Memotech or JLO 64K boards (and maybe some others, but not the Byte-Back UM).

It is necessary to add a little extra circuitry to separate out the video system from the dynamic RAM system, as Sinclair never dreamed of a ZX81 with more than 16K. Very simply, you cut the M1 not trace between the Z80 (pin 27) and the ULA (pin 10). Connect pin 2 to the Z80 side and the output on pin 8 to the ULA side. Again, use an ohm meter to chase down the A14, A15, Vcc (5 volts) and ground lines. They can all be picked up at or near the edge connector (except pin 8), but you may find some more convenient places. The diagrams show some very good places to pick up these traces. Be sure to remove all ICs (if possible) before doing any soldering.

Any relocatable programs will run here and not be affected by NEW or reset. ZXLR-8, Delphic toolkit, compiled programs with the Bob Berch Compiler, Hot Z and an assortment of other utilities will work up there and NOT interfere with any 16K program on the market.

Additional Byte-Back Memory Pack Notes:

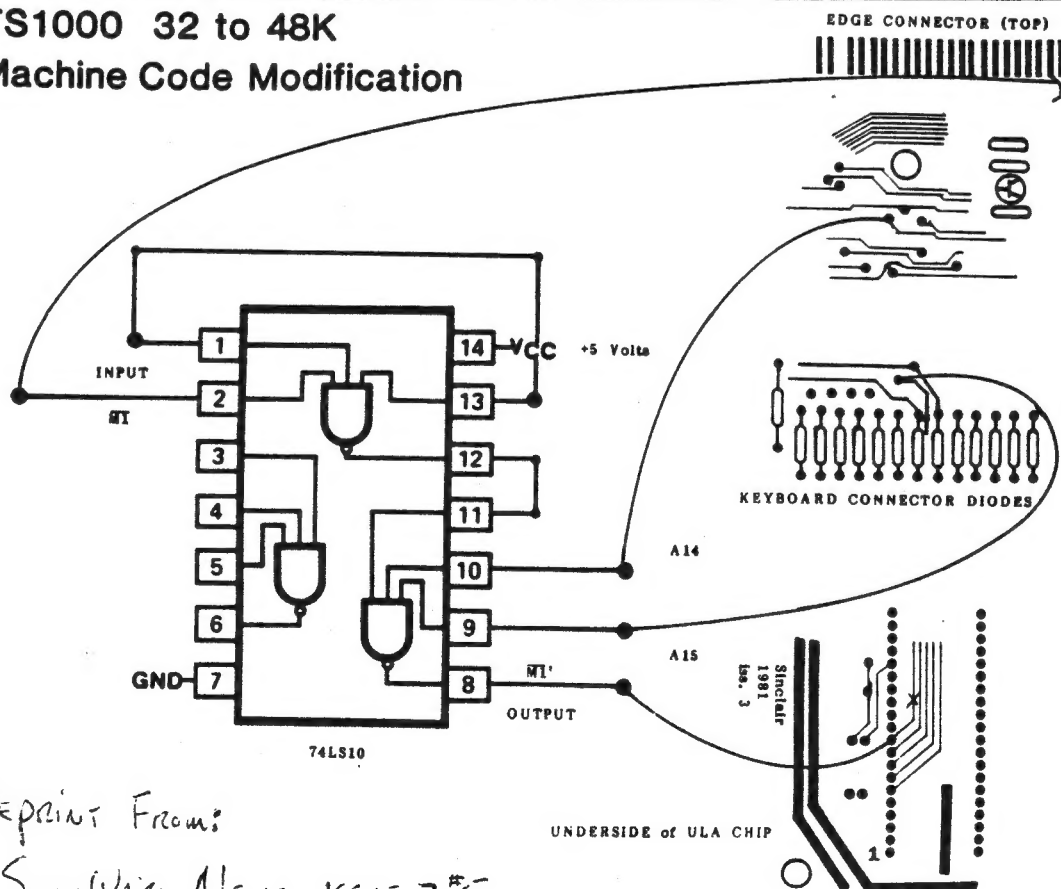
Jeffrey D. Moore
602 S. Mill Street
Louisville, Ohio 44641

Thanks to John Oliger's NOT M1 decoding circuit, many new and excellent pieces of software are coming into the market place that make use of machine language routines in the 32K-48K (8000-BFFF hex) region of memory. Some examples of these would include Hot Z-II by Ray Kingsley and Memotext in RAM, Version 3, modified and marketed by Fred Nachbaur.

However, there is one catch! It is best summed up by Ray Kingsley in his Hot Z-II User's Notes. "If you have a suitable memory...Oliger or Memotech or possibly another: Not a Byte-Back M-64 (or UM-64K memory-JM)...and you make the Oliger modification to your computer, you can run machine code in the 32K-48K block..."

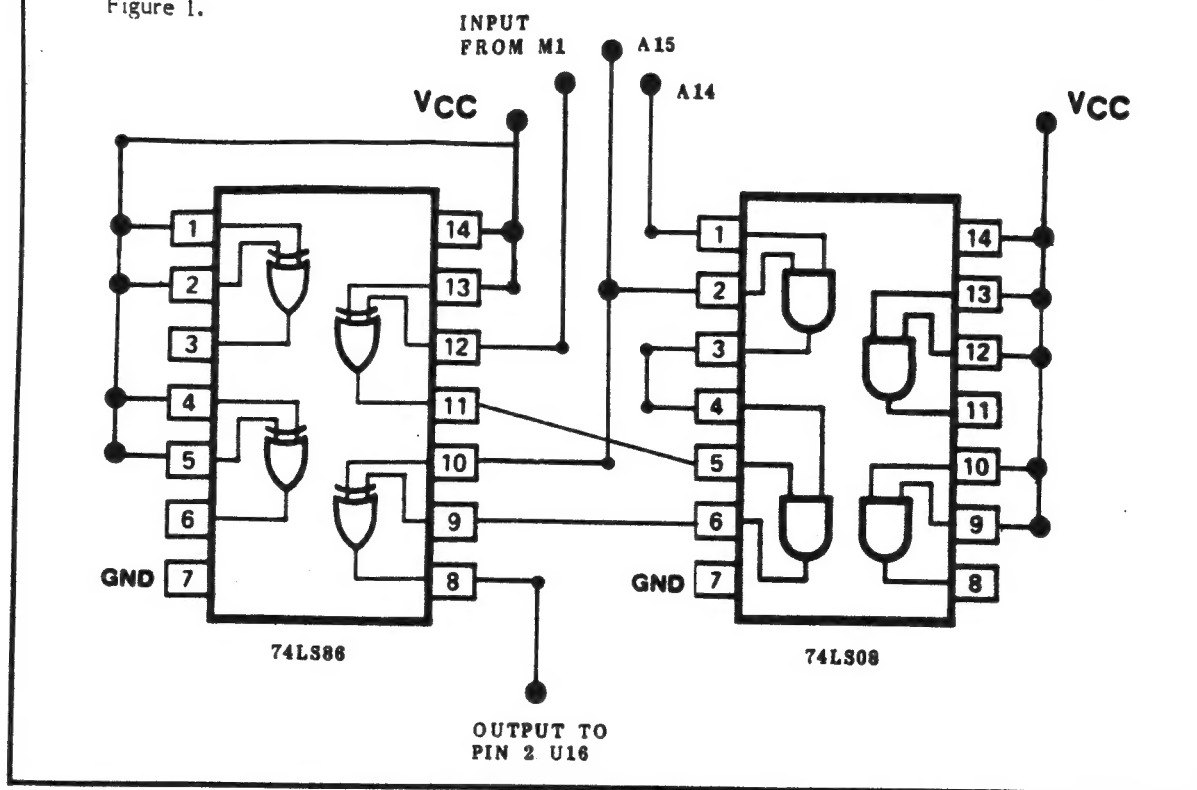
I contacted Mr. Oliger, and he sadly confirmed that it was true, a Byte-Back M-64 or a UM-64K memory would not work with the NOT M1 decoding circuit. However, our conversation, and subsequent correspondence, led Mr. Oliger to develop the two chip

TS1000 32 to 48K Machine Code Modification



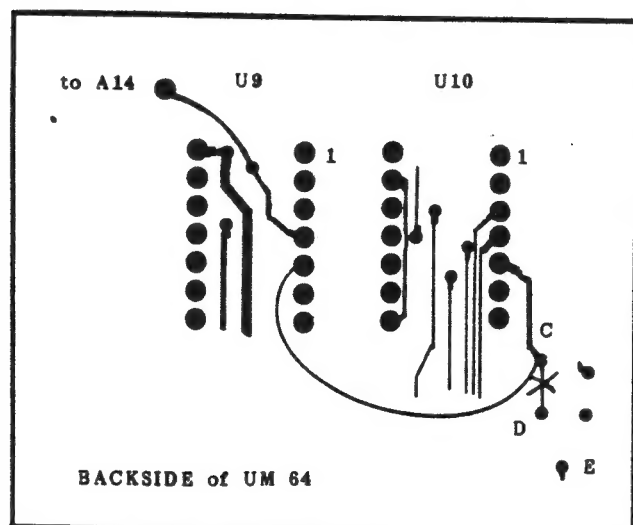
MODIFICATION for the BYTE BACK UM-64 RAM PACK

Figure 1.



buffer circuit shown in figure 1. I implemented and tested it in the Byte-Back M-64 Memory Pack, and found that it works great!

The buffer circuit itself should be constructed on a small universal circuit board (Radio Shack 276-150, or equiv., works well). Mount the buffer circuit on the heat sink side of the M-64/UM-64K memory circuit board with double-faced foam tape after all the wiring and inter-connections are complete. Care MUST be taken to fully electrically insulate the small board from the main board.



M-64 Hook-up

Modification of the M-64K memory board requires that two circuit traces be cut and seven wires installed to add the new circuit to the existing memory circuit. Find the 74LS08 (U12) chip on the M-64K memory board. Locate the circuit trace going from a feed-thru hole to pin 6 of this IC. Cut the trace between the feed-thru hole and pin 6 of U12. On the heat sink side of the M-64K memory board, locate the circuit trace going to pin 4 of the 74LS32 IC (U11) and cut it at a convenient place. Using the plated feed-thru holes on the M-64K memory board where possible as solder points (not necessary, but it makes for a neater job), make the following connections:

- 1) Connect Vcc of the buffer board to the hole for pin 16 of the spare chip on the M-64K memory board.
- 2) Connect GND (ground) of the buffer board to the hole for pin 8 of the spare chip on the M-64K memory board.
- 3) Connect A15 of the buffer board (74LS08 pin 2 and 74LS86 pin 10) to pin 5 of the 74LS08 (U12) on the M-64K memory board.
- 4) Install a jumper on the M-64K memory board from pin 5 of the 74LS08 (U12) to pin 4 of the 74LS32 (U11).
- 5) Connect A14 of the buffer board (74LS08 pin 1) to pin 5 of the 74LS32 (U11) on the M-64K memory board.
- 6) Connect NOT M1 of the buffer board (74LS86 pin 12) to pin 4 of the 74LS08 (U12) on the M-64K memory board.
- 7) Finally, connect the output of the buffer board (74LS86 pin 8) to pin 2 of the 74LS244 IC chip (U16) closest to the 74LS08 chip (U12) on the M-64K memory board.

UM-64 Connections

On the UM-64K memory board, one jumper needs to be removed, one trace needs to be cut, and seven wires must be connected. [Note: If you install the decoder board on the back of the UM-64 board, you may not have room for the backup battery. If this is important, consider mounting the decoding chips upside-down on one of the RAM IC's on the front side with crazy glue. Run "flying leads" to the appropriate circuit points. -ed.]

- 1) Remove the jumper (trace cut) going from "C" to "D".
- 2) Locate the circuit trace going to pin 2 of the 74LS244 IC (U16) closest to the EPROM socket and cut the trace in a convenient location, as close to pin 2 as possible.

Using feed-thru holes where available, make the following connections:

- 3) Connect +Vcc on the buffer board to +5 volts on the UM-64K memory board at a convenient location.
- 4) Connect GND of the buffer board to GND on the UM-64K memory board at a convenient location.
- 5) Connect A15 of the buffer board (74LS86 pin 10 and 74LS08 pin 2) to pin 5 of 74LS08 (U10) of the UM-64K memory board.
- 6) Install a jumper on the UM-64K memory board from pin 5 of the 74LS08 (U10) to "C" (A15 needs to be connected to pin 5 of U9, through "C" is the easiest way to get there).
- 7) Connect A14 of the buffer board (74LS08 pin 1) to pin 4 of the 74LS32 (U9) of the UM-64K memory board.
- 8) Connect NOT M1 of the buffer board (74LS86 pin 12) to pin 4 of the 74LS08 (U10) of the UM-64K memory board.
- 9) Finally, connect the output of the buffer board (74LS86 pin 8) to pin 2 of the 74LS244 chip (U16) closest to the PROM socket on the UM-64K memory board.

Be sure to double check all of your connections to insure that they are correct and that you have not created any solder bridges. Make sure the buffer board is secure to the M-64K memory board and that they are insulated from each other.

If you haven't already done so, make the modifications to your computer for NOT M1 Decoding per John Oliger's article and test that circuit. When all is well, power down your computer and connect the modified M-64K/UM-64K Memory Pack to it. Power back up. If that old familiar little "K" cursor shows up on the screen, you're over half way home. If not, immediately power down and re-check the buffer circuit and its connections for shorts and improper wiring. When everything checks out and powers up O.K. with the modified memory pack connected, enter the following commands: POKE 40000,201 followed by PRINT USR 40000. The screen should return 40000. This is a fair indication that everything is pretty much as it should be, but it is not foolproof. The "acid" test is to now load or write a program that runs MC in the 32K-48K region of memory and try it. It should now run correctly without any problems.